#include<iostream>

#include<conio.h>

#include<cassert>

using namespace std;

template<class Type>

struct nodeType

{

Type info;

nodeType<Type>\* link;

};

template<class Type>

class linkedQueueType

{

private:

nodeType<Type> \* queueFront;

nodeType<Type>\* queueRear;

public:

const linkedQueueType<Type>& operator=(const linkedQueueType<Type>& otherQueue)

{

nodeType<Type>\* newNode,\* current,\* last;

if (otherQueue.queueFront == NULL)

queueFront = NULL;

else

{

current = otherQueue.queueFront;

queueFront = new nodeType<Type>;

queueFront->info = current->info;

queueFront->link = NULL;

last = queueFront;

current = current->link;

while (current != NULL)

{

newNode = new nodeType<Type>;

newNode->info = current->info;

newNode->link = NULL;

last->link = newNode;

last = newNode;

current = current->link;

}

}

}

bool isEmptyQueue() const

{

return(queueFront == NULL);

}

bool isFullQueue() const

{

return false;

}

void initializeQueue()

{

nodeType<Type>\* temp;

while (queueFront != NULL)

{

temp = queueFront;

queueFront = queueFront->link;

delete temp;

}

queueRear = NULL;

}

Type front() const

{

assert(queueFront != NULL);

return queueFront->info;

}

Type back() const

{

assert(queueRear != NULL);

return queueRear->info;

}

void addQueue(const Type& queueElement)

{

nodeType<Type>\* newNode;

newNode = new nodeType<Type>;

newNode->info = queueElement;

newNode->link = NULL;

if (queueFront == NULL)

{

queueFront = newNode;

queueRear = newNode;

}

else

{

queueRear->link = newNode;

queueRear = queueRear->link;

}

}

void deleteQueue()

{

nodeType<Type>\* temp;

if (!isEmptyQueue())

{

temp = queueFront;

queueFront = queueFront->link;

delete temp;

if (queueFront == NULL)

queueRear = NULL;

}

else

cout << "Cannot remove from an empty queue" << endl;

}

linkedQueueType()

{

queueFront = NULL;

queueRear = NULL;

}

linkedQueueType(const linkedQueueType<Type>& otherQueue)

{

nodeType<Type>\* newNode,\* current,\* last;

if (otherQueue.queueFront == NULL)

queueFront = NULL;

else

{

current = otherQueue.queueFront;

queueFront = new nodeType<Type>;

queueFront->info = current->info;

queueFront->link = NULL;

last = queueFront;

current = current->link;

while (current != NULL)

{

newNode = new nodeType<Type>;

newNode->info = current->info;

newNode->link = NULL;

last->link = newNode;

last = newNode;

current = current->link;

}

}

}

~linkedQueueType()

{

initializeQueue();

}

};

void main()

{

linkedQueueType<int> queue1;

int x, y;

queue1.initializeQueue();

int total\_elements, element;

cout << "Enter the total no of elements you want: ";

cin >> total\_elements;

cout << "Enter Elements: ";

for (int i = 0; i < total\_elements; i++)

{

cin >> element;

queue1.addQueue(element);

}

linkedQueueType<int> queue2(queue1);

cout << "We copied elements of queue1 to queue 2 " << endl;

cout << " Elements OF Queue 2 are : ";

while (!queue2.isEmptyQueue())

{

cout << queue2.front() << " ";

queue2.deleteQueue();

}

cout << endl;

queue1.deleteQueue();

cout << "Queue1 Elements: ";

while (!queue1.isEmptyQueue())

{

cout << queue1.front() << " ";

queue1.deleteQueue();

}

cout << endl;

\_getch();

}